Speech Recognition: Key Word Spotting through Image Recognition

Alex Spaeth, David Harrison, Peter Braun, Salil Kanetkar, Sanjay Krishna Gouda

Objective

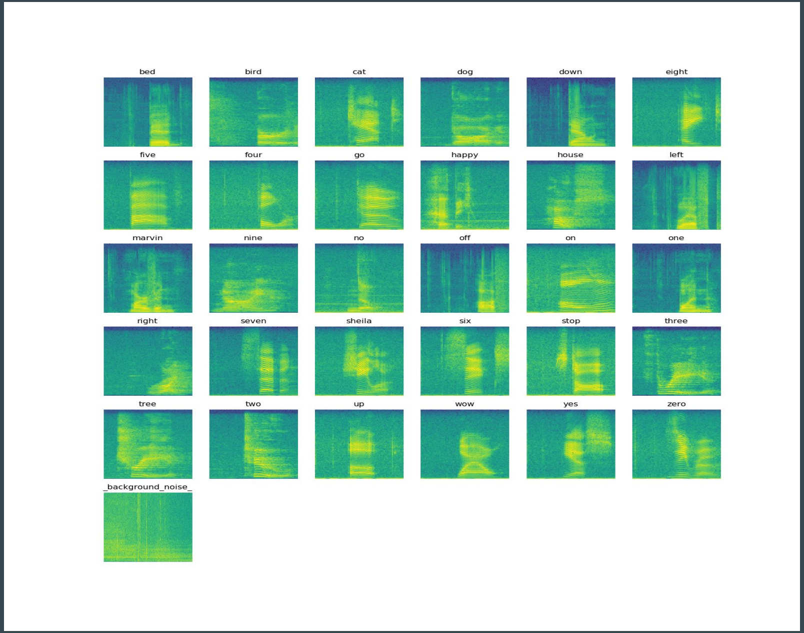
- Keyword spotting (or more simply, word spotting) is a problem that was historically first defined in the context of speech processing .
- In speech processing, keyword spotting deals with the identification of pre defined keywords in utterances.
- In this project, we will work on the task of voice command recognition, following the TensorFlow Speech Recognition Challenge that is being carried out on Kaggle.

Dataset

- Speech Commands Dataset
- 65,000 labeled audio examples
  - 1 second each
- Contains 41 different words
- 12 target labels
  - E.g. “up”, “stop”, “down”...
  - Also “<silence>” and “<unknown>”

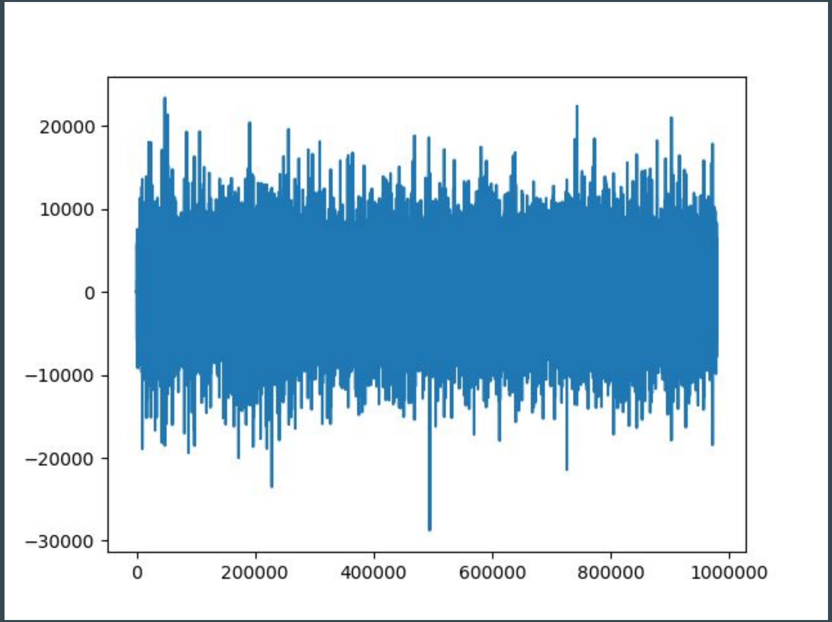
Speech to Image

- Tried 1-D CNN on raw audio with bad results, so tried to apply 2-D CNNs.
- Convert audio into meaningful images using spectrograms (a SciPy function).
- Plots of frequency content vs. time.
- Each column is the Fourier transform of the input in a window of time.
- Promising because different words actually look different! →



Speech to Image

- Use spectrograms because a plot of amplitude vs. time is uninformative →
- Can only recognize volume envelope.
- Don’t expect good performance, but we did try this type of image in one model for comparison.
- Spectrograms structure the information better, allowing analysis in terms of pitch and time rather than absolute sound pressure.



Models - Initial

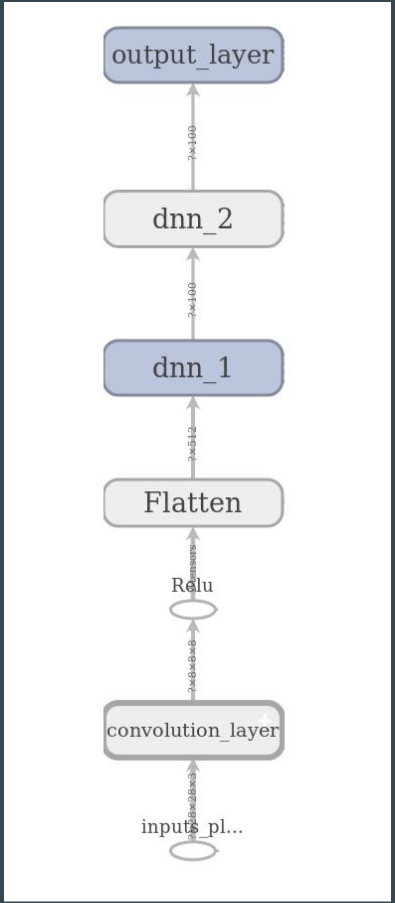
a CNN with two pooling convolutional layers and one FC layer.

- the model suggested by the authors of the challenge
- a large model: about 800 million FLOPs per inference, and 940,000 weights
- good performance (89% validation), but convergence takes > 8h
- compared some more computationally tractable models

Models

Low latency convolutional Model:

- Convolution Layer:
  - Filter: 7x7
  - Stride: 3
- Dnn1: 100 units (no nonlinearity)
- dnn2 : 100 units (relu)
- 63,800 Parameters
- Estimated: 11 million FLOPs (7 times less)



Results

Low Latency Convolutional Model

Xavier Initialization:

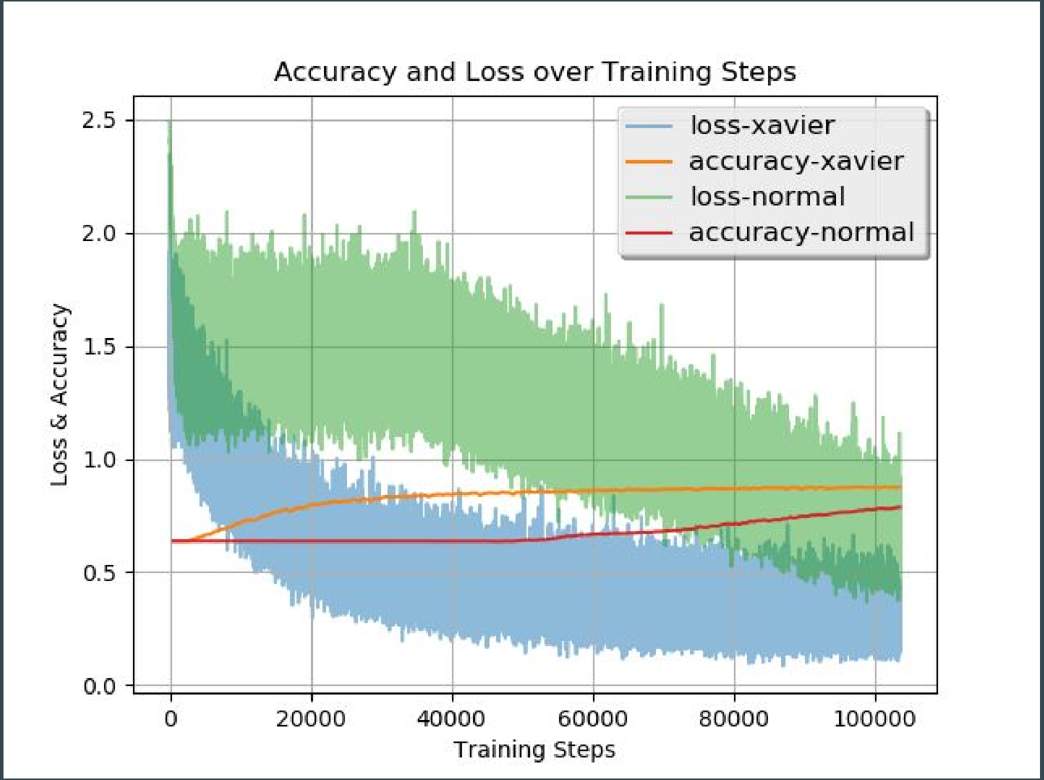
- Matrix **M** (n by m)
- Assign values uniformly sampled from [-ϵ, ϵ]
- ϵ = √6 / √(m + n)

Best Accuracies:

- Xavier: 87.6%, epoch 196
- Truncated Normal: 78.7%, epoch 200

We also tried constant initialization methods

- all weights = 0.5
- did not converge

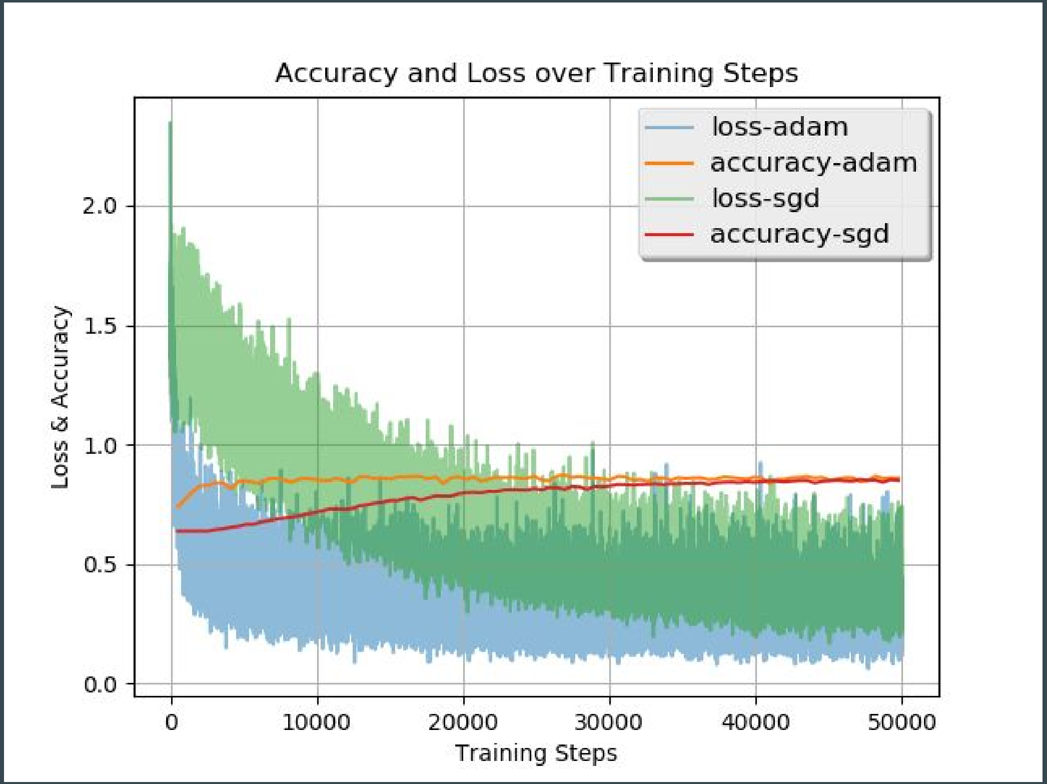


Results

Low Latency Convolutional Model

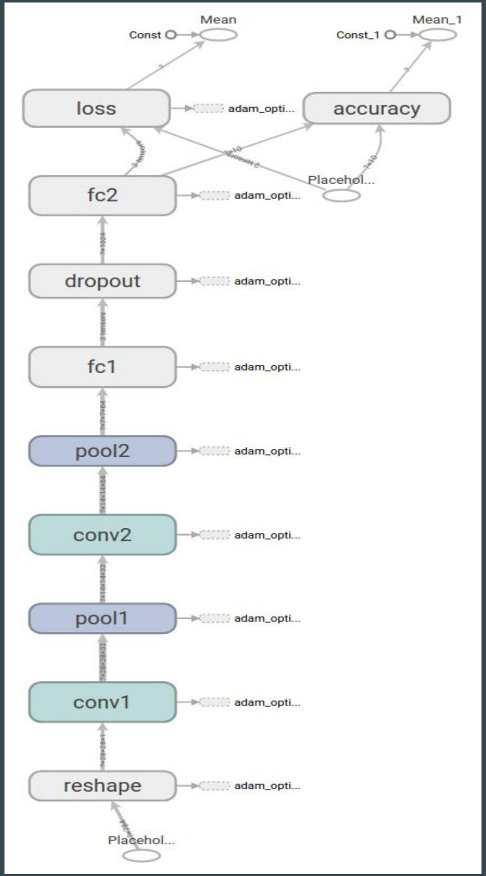
Optimization Algorithms:

- Adam: adaptive learning rate
  - Best accuracy, epoch: 87.1%, 52
- SGD
  - Best accuracy, epoch: 87.6%, 196



Tensorflow MNIST expert

- We also tested the deep CNN available on the Tensorflow website
- It has two convolutional layers, 2 pooling layers and a fully connected layer
- It also employs dropout to combat overfitting and uses ReLu as the activation function.



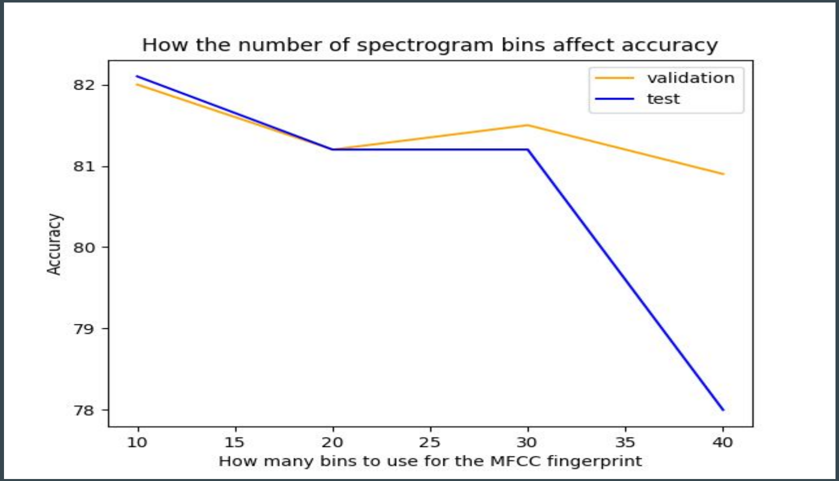
Models

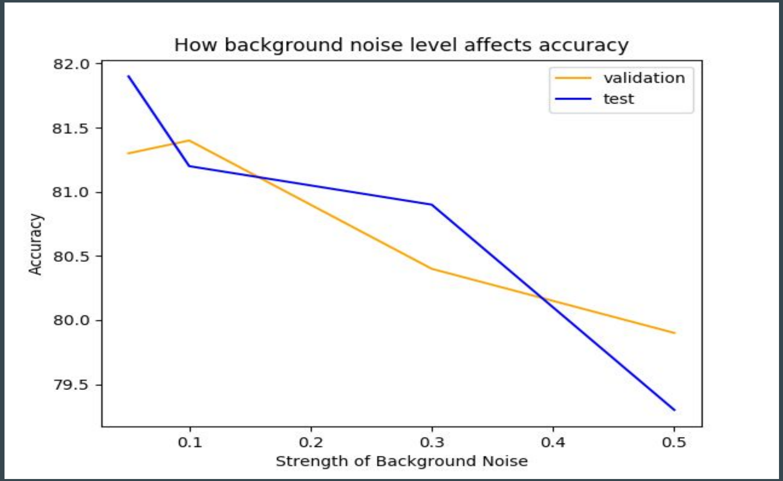
MNIST CNN

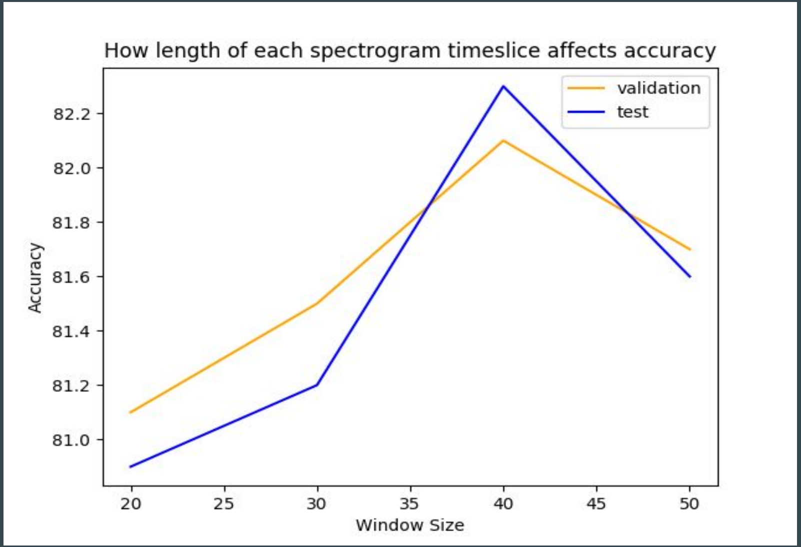
- We used the MNIST sample CNN architecture for this process.
- It is a simple, neural network with two convolutional layers and two maxpool layers. The architecture is similar to the default model, but much smaller.

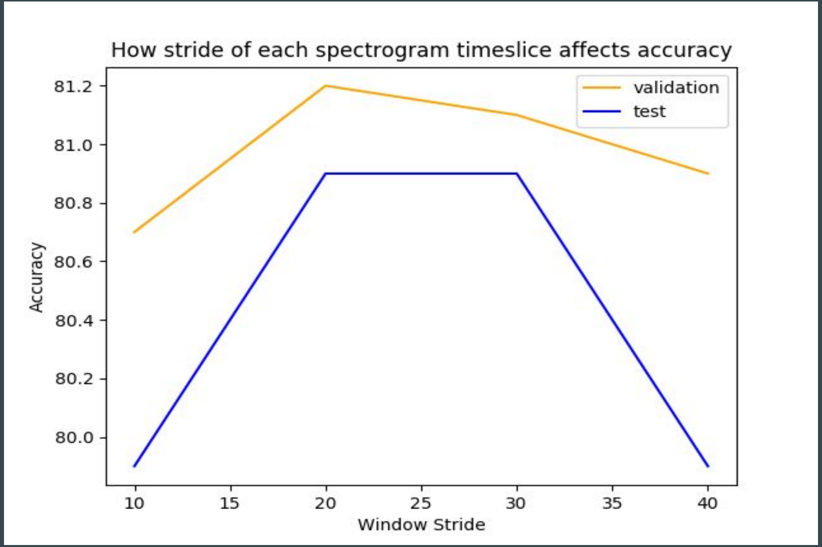
Results

- Varying hyperparameters on Low Latency Convolutional Model






How length of each spectrogram timeslice affects accuracy
validation
test
Accuracy
Window Size
82.2
82.0
81.8
81.6
81.4
81.2
81.0
20
25
30
35
40
45
50


How stride of each spectrogram timeslice affects accuracy
validation
test
Accuracy
Window Stride
81.2
81.0
80.8
80.6
80.4
80.2
80.0
10
15
20
25
30
35
40

also tried a CNN intended for MNIST handwriting classification

- on log spectrograms of the audio files: validation accuracy 52.7%
- on time vs. amplitude images: validation accuracy 50.3%

Model - CNN

A 5-layer CNN and a 7-layer CNN.

Two variations of the bigger architecture. Different number of filters.

Two training variants - vanilla CNN and CNN with adversarial training.

Architecture:

- Group of convolutions using Convolutional layer followed by max pooling layer.
- Flatten.
- Fully Connected.
- Dropout? Further.

CNN - Architecture

Idea from MCDNN.

(#2 for MNIST)

Smaller CNN :

Input → (convolutional layer → relu+dropout → max pooling) x3 → flatten → fully connected → fully connected + dropout.

Larger CNN :

Input → (convolutional layer → elu → max pooling) x 5 → fully connected → fully connected → optional dropout.

Also differ in number of filters used. Details in report (too many to include in here).

Adversarial Training

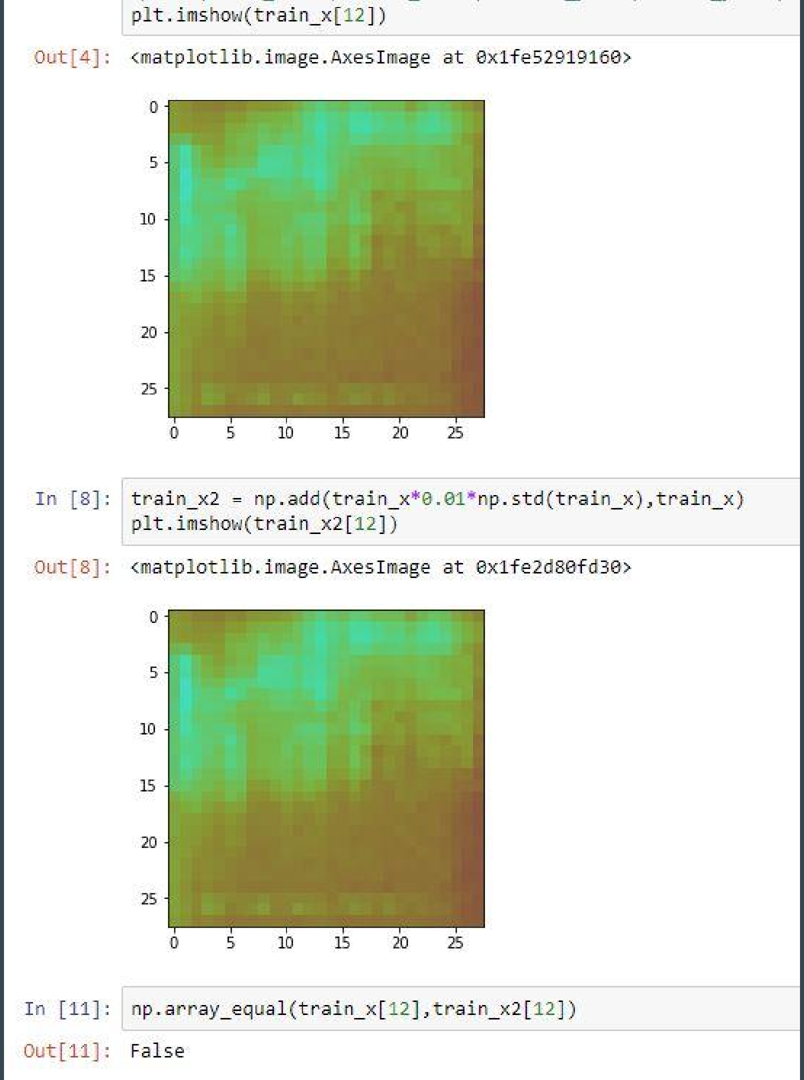
Followed Szegedy et al [2012] and Mikolov (VAT) to add adversarial training.

Method:

Augment more data (images) that are “visually similar” to existing ones.

Why:

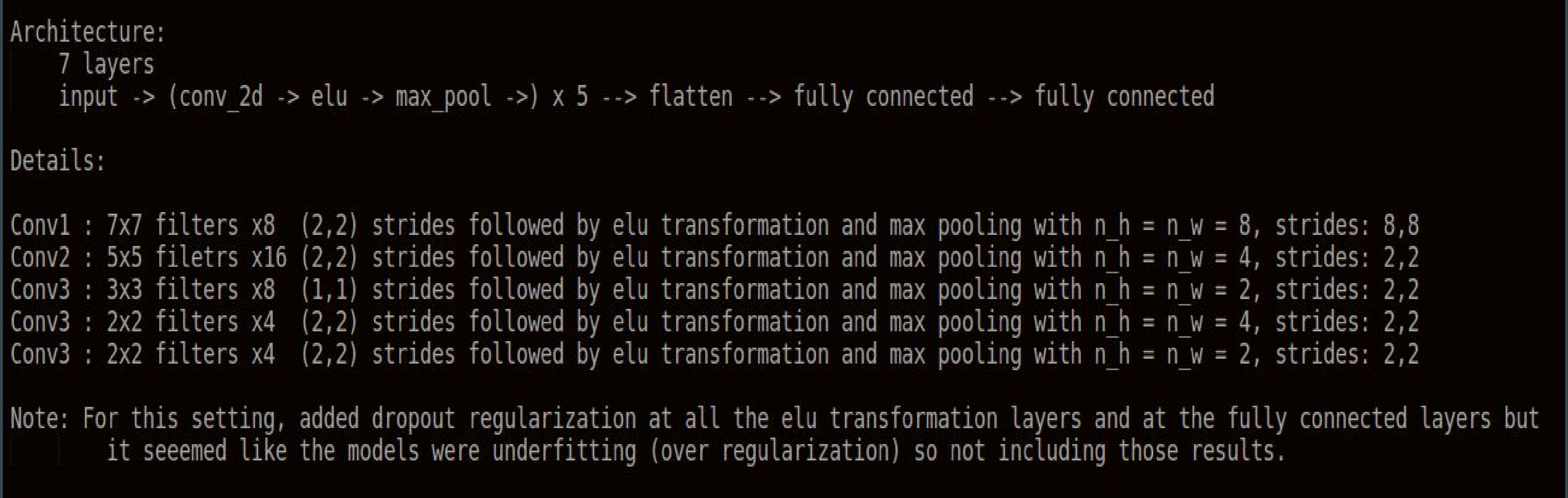
Adds regularization.



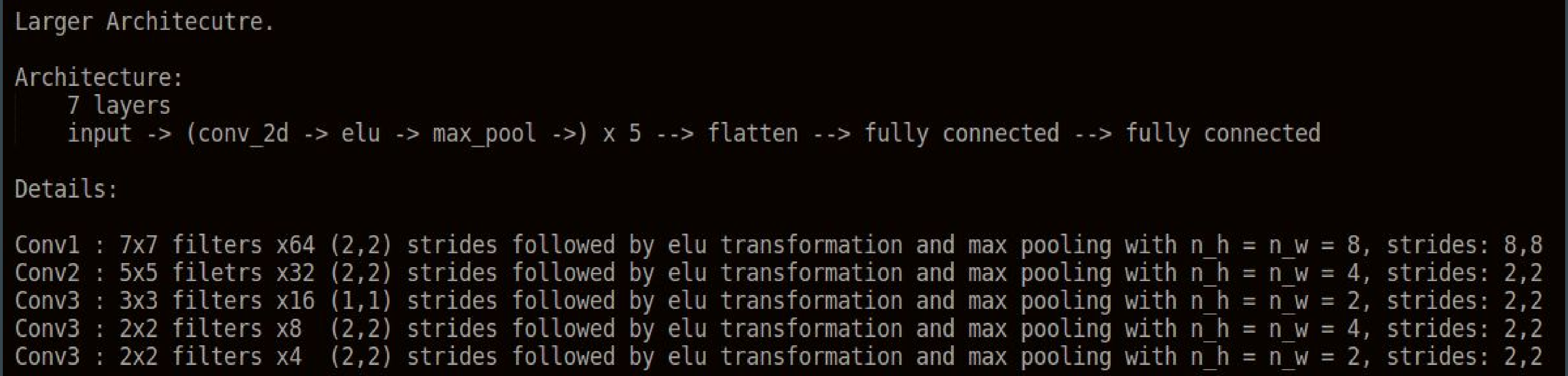
```
         plt.imshow(train_x[12])

Out[4]:  <matplotlib.image.AxesImage at 0x1fe52919160>
```

```
In [8]:  train_x2 = np.add(train_x*0.01*np.std(train_x),train_x)
         plt.imshow(train_x2[12])

Out[8]:  <matplotlib.image.AxesImage at 0x1fe2d80fd30>
```

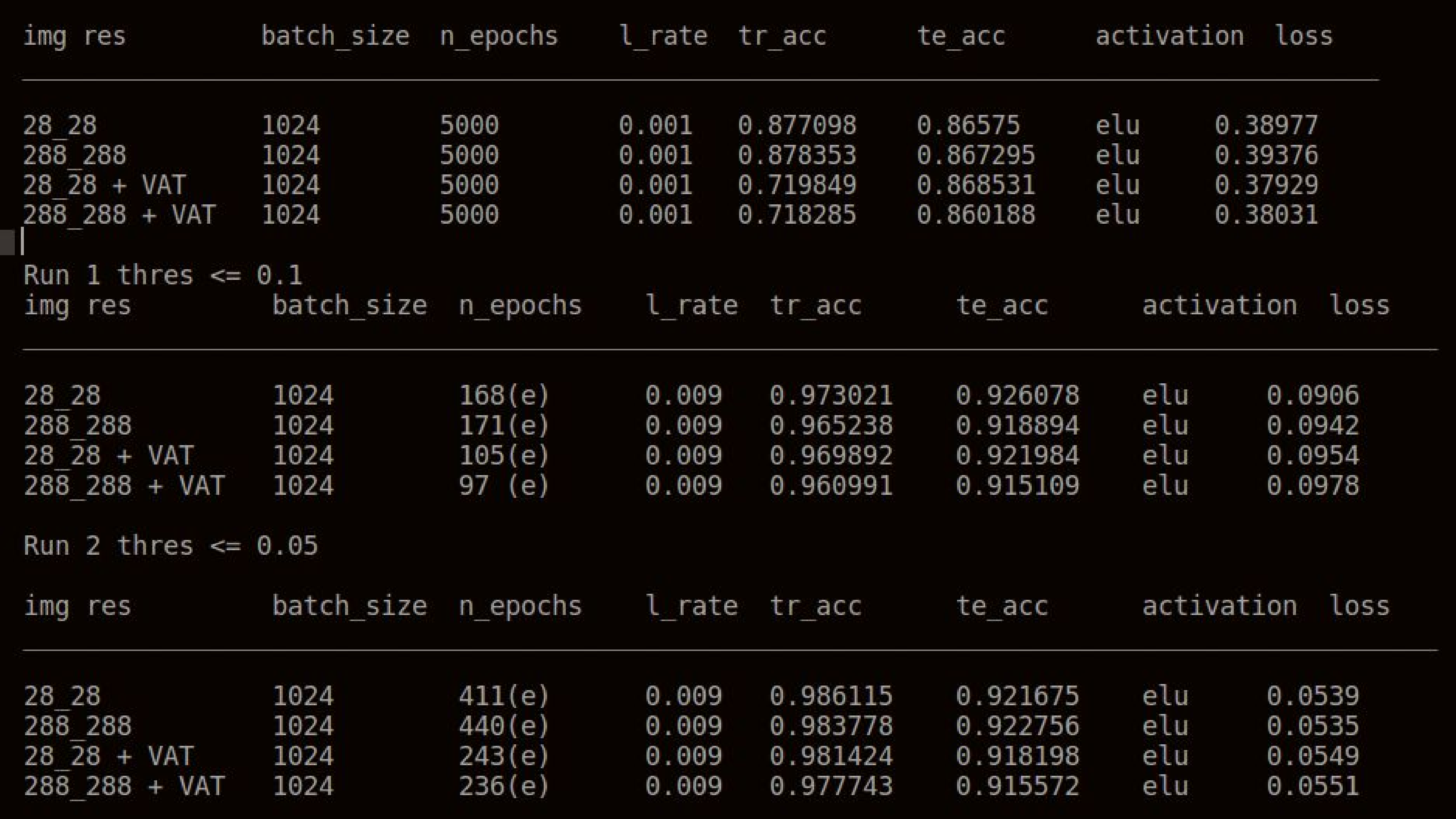
```
In [11]:  np.array_equal(train_x[12],train_x2[12])

Out[11]:  False
```



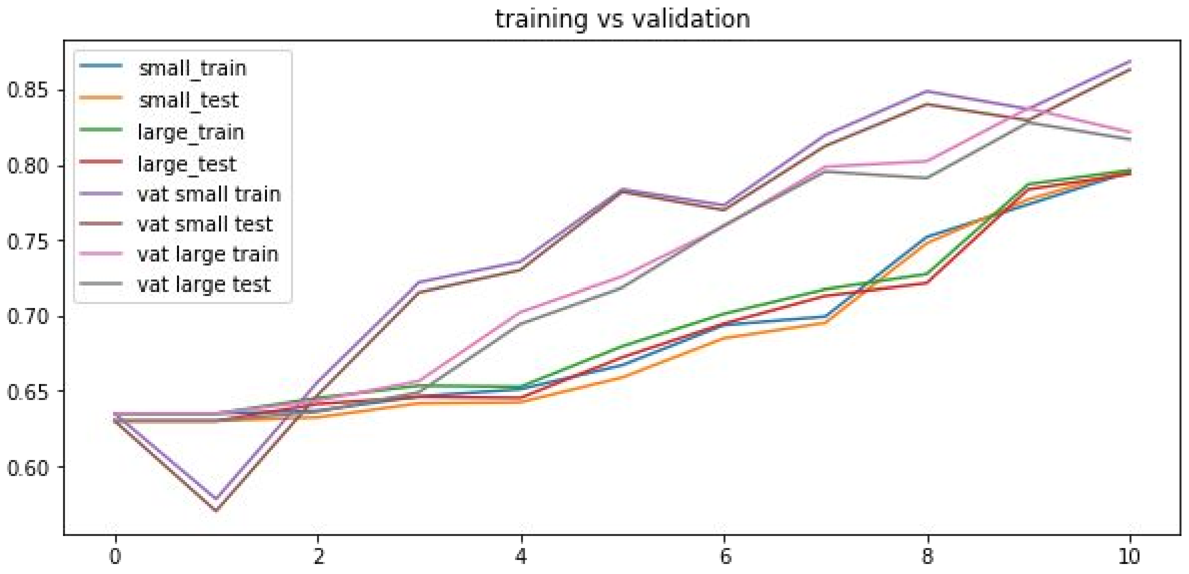
```
Architecture:
    7 layers
    input -> (conv_2d -> elu -> max_pool ->) x 5 --> flatten --> fully connected --> fully connected

Details:

Conv1 : 7x7 filters x8  (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 8, strides: 8,8
Conv2 : 5x5 filetrs x16 (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 4, strides: 2,2
Conv3 : 3x3 filters x8  (1,1) strides followed by elu transformation and max pooling with n_h = n_w = 2, strides: 2,2
Conv3 : 2x2 filters x4  (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 4, strides: 2,2
Conv3 : 2x2 filters x4  (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 2, strides: 2,2

Note: For this setting, added dropout regularization at all the elu transformation layers and at the fully connected layers but
        it seeemed like the models were underfitting (over regularization) so not including those results.
```

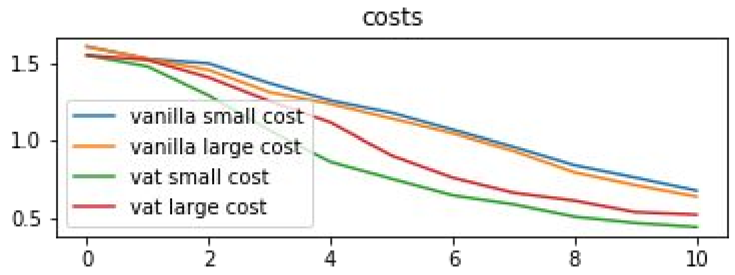


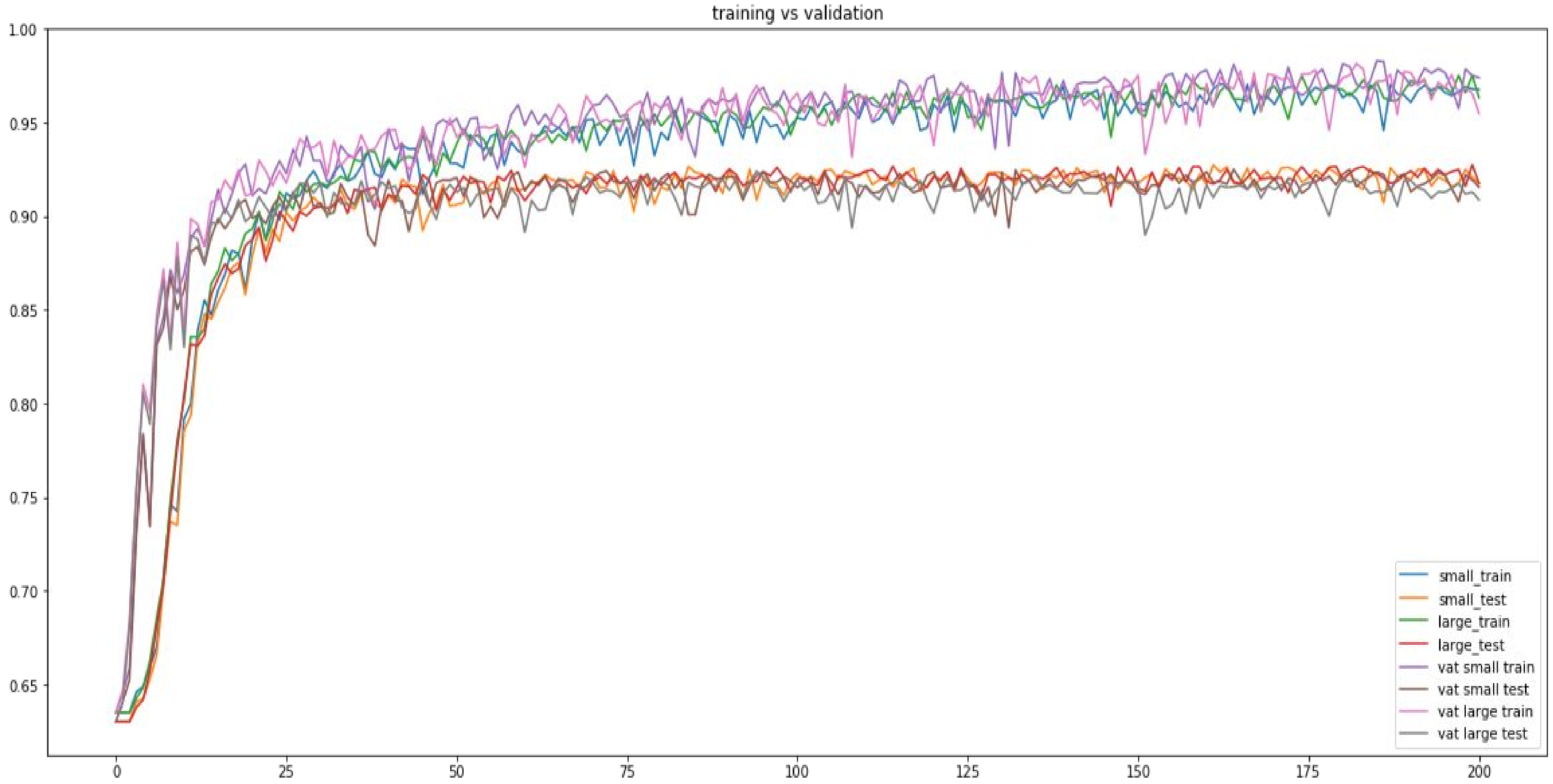
```
Larger Architecutre.

Architecture:
    7 layers
    input -> (conv_2d -> elu -> max_pool ->) x 5 --> flatten --> fully connected --> fully connected

Details:

Conv1 : 7x7 filters x64 (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 8, strides: 8,8
Conv2 : 5x5 filetrs x32 (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 4, strides: 2,2
Conv3 : 3x3 filters x16 (1,1) strides followed by elu transformation and max pooling with n_h = n_w = 2, strides: 2,2
Conv3 : 2x2 filters x8  (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 4, strides: 2,2
Conv3 : 2x2 filters x4  (2,2) strides followed by elu transformation and max pooling with n_h = n_w = 2, strides: 2,2
```

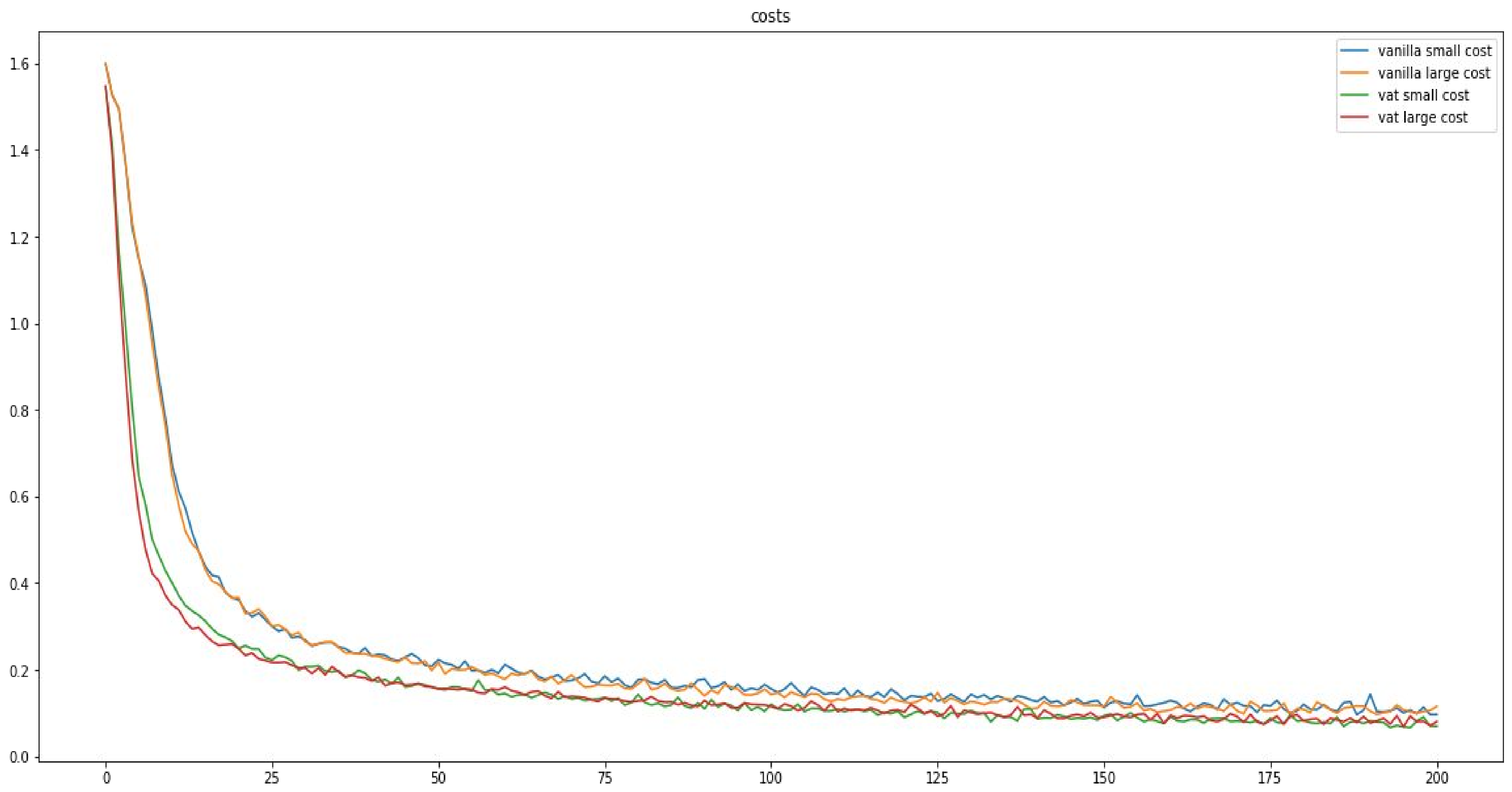


```
img res          batch_size  n_epochs    l_rate  tr_acc      te_acc      activation  loss
_____________________________________________________________________________________

28_28            1024        5000        0.001   0.877098    0.86575     elu      0.38977
288_288          1024        5000        0.001   0.878353    0.867295    elu      0.39376
28_28 + VAT      1024        5000        0.001   0.719849    0.868531    elu      0.37929
288_288 + VAT    1024        5000        0.001   0.718285    0.860188    elu      0.38031

Run 1 thres <= 0.1
img res          batch_size  n_epochs    l_rate  tr_acc      te_acc      activation  loss
_____________________________________________________________________________________

28_28            1024        168(e)      0.009   0.973021    0.926078    elu      0.0906
288_288          1024        171(e)      0.009   0.965238    0.918894    elu      0.0942
28_28 + VAT      1024        105(e)      0.009   0.969892    0.921984    elu      0.0954
288_288 + VAT    1024        97 (e)      0.009   0.960991    0.915109    elu      0.0978

Run 2 thres <= 0.05

img res          batch_size  n_epochs    l_rate  tr_acc      te_acc      activation  loss
_____________________________________________________________________________________

28_28            1024        411(e)      0.009   0.986115    0.921675    elu      0.0539
288_288          1024        440(e)      0.009   0.983778    0.922756    elu      0.0535
28_28 + VAT      1024        243(e)      0.009   0.981424    0.918198    elu      0.0549
288_288 + VAT    1024        236(e)      0.009   0.977743    0.915572    elu      0.0551
```


training vs validation
small_train
small_test
large_train
large_test
vat small train
vat small test
vat large train
vat large test
0.85
0.80
0.75
0.70
0.65
0.60
0
2
4
6
8
10


costs
vanilla small cost
vanilla large cost
vat small cost
vat large cost
1.5
1.0
0.5
0
2
4
6
8
10


training vs validation
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
25
50
75
100
125
150
175
200
small_train
small_test
large_train
large_test
vat small train
vat small test
vat large train
vat large test


costs
vanilla small cost
vanilla large cost
vat small cost
vat large cost
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200